Exercise 05 — Operating the Cluster

Ansible for Proxmox VE · Module 05

credativ GmbH

~20 min · in `~/ansible-proxmox` · needs a working cluster from exercise 03

> **Objective:** give the cluster storage, a service account, and a guest that installs itself — all through the API, from one playbook.

> **i** Where to look
>
> Everything here was on the slides. The handout has each task in full, with the `curl` equivalent and a link into the API viewer of your own cluster (`https://192.168.0.1:8006/pve-docs/api-viewer/`). `proxmox_storage` · `proxmox_user` · `proxmox_kvm` · `proxmox_backup` · `blockinfile`

`playbooks/30-operations.yml` is a skeleton again: the play structure, the `module_defaults` block and every task name are there, each body a `TODO`. `inventory/group_vars/all.yml` already holds the values.

1 — Check the ground before you build on it

The variables assume things about your cluster. Verify the one that actually varies:

```
ansible pve01 -m ansible.builtin.command -a 'pvesm status'
```

Done when: the `Name` column contains the storage named in `pve_vm_storage`. On this lab that is `local-lvm`; on a ZFS install it would be `local-zfs`.

2 — Storage, in two halves

Two tasks, two different worlds — that is the lesson of this one:

- the backup **directory** on the nodes: `file`, over SSH, on every host
- the same directory **registered as cluster storage**: `proxmox_storage`, `type: dir`, one API call for the whole cluster
- and the shared **NFS library**: `proxmox_storage`, `type: nfs`, `content: [import, iso, vztmpl]`

```
ansible-doc community.proxmox.proxmox_storage
```

Done when: the three task bodies no longer say `TODO`.

3 — A service account with a token

Three tasks: the group, the user with a token, and the permission on `/`.

> **!** Important
>
> `privsep: false` gives the token its user's permissions. With `privsep: true` the token additionally needs its own ACL, and the effective rights are the **intersection** of both — a token that works perfectly for one call and returns an empty list for the next is almost always this.

The API returns a token secret exactly once, at creation. The playbook writes it to `~/pve-token.json`; module 06 needs it.

Done when: those three bodies no longer say `TODO`.

4 — A guest that installs itself

Four tasks: activate the NFS library and find the image, create the VM from it, start it, wait for SSH.

```
ansible-doc community.proxmox.proxmox_kvm   # scsi, ide, net, boot, sshkeys, timeout
```

Read the comment above the first of them before you write it. Proxmox VE mounts an NFS storage on **first activation**, not when you attach it, so importing in the same run fails with `can't find file` — and waiting for the path does not help, because nothing has asked the node to mount it yet. Listing the contents activates the storage and confirms the image in one go.

Then add the backup task after it.

Done when: the bodies are filled in. Do not run it yet.

5 — The direct route, then run it

The last play edits `/etc/network/interfaces` on the node with `blockinfile` and applies it with `ifreload -a`. No module owns this file.

> **⚠** Warning
>
> `ifreload -a` applies the change **immediately**. Get the block wrong and you can cut the node off the network. That is why the playbook keeps a pristine copy first, and why the marker comment matters: `blockinfile` uses it to find its own block again instead of appending a second one on every run.

Now run the whole thing, twice:

```
ansible-playbook playbooks/30-operations.yml
ansible-playbook playbooks/30-operations.yml
```

Done when: the first run looks like this, and the second is quiet apart from one task:

```
# first run
localhost : ok=11  changed=9  unreachable=0  failed=0
pve01     : ok=1   changed=1  unreachable=0  failed=0
pve02     : ok=1   changed=1  unreachable=0  failed=0
pve03     : ok=4   changed=4  unreachable=0  failed=0

# second run
localhost : ok=10  changed=1  unreachable=0  failed=0  skipped=1
```

That `changed=1` is **Back up the VM**, and it will never be anything else: a backup is an action, not a state, so there is nothing for the module to compare against. Which task is skipped on the second run, and why? The answer is in the playbook.

`web01` is running in the GUI, and `ssh student@192.168.0.120` gets you in without a password.

If you finish early

- Look at `~/pve-token.json`. Why can the playbook only write it once?
- Open the API viewer and find `POST /nodes/{node}/qemu`. Compare its parameter names with the options you passed to `proxmox_kvm`.
- Create a container instead: `community.proxmox.proxmox` with `pve_ct_template`.
- Add a second guest by changing `pve_vm_id` and `pve_vm_name` on the command line with `--extra-vars`.
- Scheduled backup jobs have no module. Find the API path that would create one.

Solutions

Solution — Task 1: check the ground

```yaml
# inventory/group_vars/all.yml
pve_backup_path: /var/lib/vz-backup
pve_vm_storage: local-lvm
pve_vm_node: pve02
pve_vm_id: 9001
pve_vm_name: web01
```

Verify the storage name against the real environment before the session:

```
ansible pve01 -m ansible.builtin.command -a 'pvesm status'
```

On an installation with ZFS root there is no `local-lvm`; the equivalent is `local-zfs`. On a very small nested lab disk there may be no thin pool at all, in which case use `local` with `format: qcow2`. Set the value in the workshop material before you deliver it, do not let twelve participants debug it live.

Solution — Task 2: storage, both halves

The playbook is printed in the task; reference copy `courses/ansible-proxmox/code/playbooks/30-operations.yml`.

Verify:

```
ansible pve01 -m ansible.builtin.command -a 'pvesm status'
# pve-backup   dir   active ...
ansible pve -m ansible.builtin.command -a 'ls -ld /var/lib/vz-backup'
```

Errors to expect:

- `module_defaults` indented under `tasks:` instead of the play → the modules complain about missing `api_host`.
- Storage registered before the directory exists → Proxmox VE marks it inactive. The play order in the solution prevents this; if someone reversed the plays, the fix is to re-run, which is a nice demonstration of convergence.
- `content: [backup]` forgotten → task 5 fails later with a message about the storage not supporting the content type. Worth letting one participant hit it.

Solution — Task 2, continued: the NFS library

Task as printed.

Verify:

```
ansible pve -m ansible.builtin.command -a 'pvesm status'
#   nfs-shared    nfs    active ...    on all three nodes
ansible pve01 -m ansible.builtin.command -a 'ls /mnt/pve/nfs-shared/import/'
```

Points to make:

- One API call mounted the share on every node. Compare with the previous task, where the directory had to be created on each node over SSH first. Shared storage is a cluster object; local storage is not.
- `content: [import, iso, vztmpl]` has no `images` and no `rootdir`, so Proxmox VE will never place a guest disk there. On a library that several clusters read from, that is the difference between a catalogue and a landfill.
- The export is restricted to the training network, and the nodes reach it through the desktop. If it fails, check routing before you check Proxmox VE.

If it fails: `mount.nfs: access denied` means the node's address is not in the export list. `active: 0` in `pvesm status` with no error usually means the server is unreachable rather than misconfigured.

Solution — the SSH key for the guest

```
ssh-keygen -t ed25519 -N "" -f ~/.ssh/id_ed25519
```

Participants who already have a key can skip this. The `lookup` in the next task reads `~/.ssh/id_ed25519.pub` on the **control node**, so a key in any other location needs the path adjusting.

Solution — Task 3: the service account

Tasks as printed. The two teaching points:

`privsep: false`. `true` is the default, in the API and in the module. A separated token's effective permissions are the intersection of the user's and the token's own ACL, so a fresh one can do nothing at all until the token itself is granted something — regardless of what its user may do. Show it if there is time: set `privsep: true`, re-run task 4, read the error. Then note that reads behave differently and come back empty rather than failing, which is the version that actually wastes people's time. The handout has the full rule and an ACL example for `type: token`.

The secret appears once. `automation_user.secrets` is populated only on the run that creates the token. On every later run it is absent, which is why the file task carries `when: automation_user.secrets is defined`. If a participant loses the secret, the recovery is to delete and recreate the token:

```
ansible pve01 -m ansible.builtin.command -a 'pveum user token remove
automation@pve ansible'
```

Verify:

```
ansible pve01 -m ansible.builtin.command -a 'pveum user list'
ansible pve01 -m ansible.builtin.command -a 'pveum user token list
automation@pve'
ansible pve01 -m ansible.builtin.command -a 'pveum acl list'
cat ~/pve-token.json      # mode 0600
```

Solution — Task 4: the guest

Task as printed. Timings on this lab: import a few seconds, boot and cloud-init together under a minute. The whole `30-operations.yml` run takes about half a minute.

Verify:

```
ansible pve02 -m ansible.builtin.command -a 'qm config 9001'
#   scsi0: local-lvm:vm-9001-disk-0,size=3G
#   ide2: local-lvm:vm-9001-cloudinit,media=cdrom
#   ipconfig0: ip=192.168.0.120/24,gw=192.168.0.254
ssh student@192.168.0.120 'hostname; . /etc/os-release && echo $PRETTY_NAME'
```

The four parts worth explaining, in this order:

1. **`scsi0: "<storage>:0,import-from=<path>"`** — the `0` is the size, and `0` means "take it from the source image". Proxmox VE copies and converts the qcow2 into a real disk on `local-lvm`.
2. **`ide2: "<storage>:cloudinit"`** — a generated ISO holding the configuration. Leave it out and the guest boots the image unchanged, ignoring everything below.

3. **`sshkeys`** — expect someone to try `cipassword` alone and fail to log in. The Debian cloud image ships `PasswordAuthentication no`; a password-only guest is reachable on the console and nowhere else. This is a good failure to let happen.
4. **`ipconfig0`** — `ip=dhcp` works too where a DHCP server exists.

Errors to expect:

- `can't find file '/mnt/pve/nfs-shared/import/…'` although the file is plainly there when you `ls` it → the activation task was skipped. Proxmox VE mounts NFS storages lazily, so nothing had mounted the share at import time. This one costs people ten minutes because the evidence contradicts the error, so name it before they hit it.
- `Reached timeout while waiting for creating VM` → `timeout: 300` missing. The module's default is 30 seconds, and reading 326 MB over NFS takes longer.
- `unable to parse volume filename` → the storage prefix is missing from `scsi0`.
- `lookup file not found` → no key at `~/.ssh/id_ed25519.pub`, task 4 skipped.
- SSH refused after the playbook says it finished → the guest is up but cloud-init is still running. The `wait_for` covers this; without it participants race the boot.

The point of the whole task: nobody installed an operating system. The image arrived installed, and cloud-init did the last mile. That is how guests are built today, and it is why the template-plus-cloud-init pattern shows up in every serious Proxmox VE automation.

Solution — Task 4, continued: the backup

Task as printed. The backup of a 1 GiB empty disk takes a few seconds.

Verify:

```
ansible pve02 -m ansible.builtin.command -a 'ls -lh /var/lib/vz-backup/dump/'
# vzdump-qemu-9001-YYYY_MM_DD-HH_MM_SS.vma.zst
```

If it fails: the storage does not allow `backup` content (task 2), the directory does not exist on the node the VM runs on (task 2, first play), or `wait_timeout` was too short on a slow nested lab. Raise it instead of removing `wait`.

Solution — Task 5: editing the file directly

Play as printed. This is the one task in the workshop that deliberately does the thing we spent three modules avoiding. Participants should have written it once, with their eyes open.

Why `hosts: pve03` and not `pve`. A networking mistake applied to all three nodes at once ends the workshop. One node, one blast radius. State this explicitly: it is an operational habit, not a lab convenience.

Verify:

```
ansible pve03 -m ansible.builtin.command -a 'ip -br link show vmbr1'
# vmbr1   DOWN   <NO-CARRIER,BROADCAST,MULTICAST,UP>
ansible pve03 -m ansible.builtin.command -a 'cat /etc/network/interfaces'
ansible pve03 -m ansible.builtin.command -a 'ls -l /etc/network/
interfaces.orig'
```

`UNKNOWN` is correct: a bridge with no ports has no carrier state to report. Participants read this as a failure, so pre-empt it.

Points worth making while they read their own play:

- `force: false` on the copy makes the safety net idempotent. Without it, the second run overwrites the pristine copy with the already-modified file, which is exactly when you need it and exactly when it is worthless.
- `validate:` runs the command against a temporary file *before* the real file is replaced, with `%s` standing in for the path. The check used here is minimal; in production, use dedicated syntax validators: `visudo -cf %s`, `nginx -t -c %s`, `named-checkconf %s`.
- The handler runs once, after all tasks. On a real change to `vmbr0` this is the moment the session dies. That is the argument for `--check --diff` and `--limit`, not an argument against automation.

The API comparison. After running the `proxmox_node_network` task, *pve03* → *System* → *Network* shows `vmbr2` as pending while `vmbr1` is not marked at all. Both edits end in the same configuration file; the API route goes through `/etc/network/interfaces.new` and is therefore reviewable and revertible in the GUI before it takes effect. The direct edit has already happened.

Make the comparison explicit on the whiteboard:

	direct edit	`proxmox_node_network`
Idempotent	only because `blockinfile` is	yes
Reviewable before it applies	no	yes, as pending changes
Needs the collection	no	yes
Covers exotic cases	anything you can write	what the module models

Clean-up (if you want the environment tidy for the next session):

```
ansible pve03 -m ansible.builtin.copy \
      -a   'src=/etc/network/interfaces.orig   dest=/etc/network/interfaces
remote_src=true'
ansible pve03 -m ansible.builtin.command -a 'ifreload -a'
```

Solution — Task 5, continued: run it twice

On the second run:

Task	Result	Correct?
directory, storage, group, user, ACL, VM	`ok`	yes, all idempotent
token secret file	`skipped`	yes, `secrets` is only returned at creation

Task	Result	Correct?
`/etc/network/interfaces` block	`ok`	yes, `blockinfile` recognises its marker
`ifreload -a` handler	not run	yes, handlers only fire when notified
backup	`changed`	yes, a backup is an *action*, not a state

Spend a minute on this table. Idempotency applies to *declarative* modules. A backup, a reboot and a migration are events: re-running them does something again, and that is correct behaviour, not a bug. In a real playbook you would guard the backup with `when:` or move it into its own playbook that is run on purpose.

Solution — Optional tasks

Put the secret from `~/pve-token.json` into `pve_api_token_secret` in `inventory/group_vars/pve.yml`, then swap the credentials in `module_defaults`:

```yaml
  module_defaults:
    group/community.proxmox.proxmox:
      api_host: "{{ hostvars['pve01'].ansible_host }}"
      api_user: automation@pve
      api_token_id: ansible
      api_token_secret: "{{ pve_api_token_secret }}"
      validate_certs: false
```

```yaml
- community.proxmox.proxmox_pool:
    poolid: web
    comment: Web tier

- community.proxmox.proxmox_pool_member:
    poolid: web
    member: "{{ pve_vm_id }}"
    type: vm

- community.proxmox.proxmox_role:
    roleid: BackupOperator
    privs: [VM.Backup, VM.Audit, Datastore.AllocateSpace]

- community.proxmox.proxmox_access_acl:
    path: /storage/pve-backup
    type: group
    ugid: automation
    roleid: BackupOperator
    propagate: 1
```

Note for the room: the token play proves the ACL actually works. If `PVEVMAdmin` on `/` had not been granted, the VM task would now fail. The switch from root to a scoped token is the first real test of the permission model.

Automated verification

```
./tests/run.sh 05
```

Checks: storage `pve-backup` exists, is active on all nodes and allows `backup` content; the group, user, token and ACL entry exist; VM 9001 exists on the expected node with the expected disk; at least one backup archive for VMID 9001 is present; and a repeated run changes nothing except the backup task.